www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 6 Issue 6 June 2017, Page No. 21663-21678
Index Copernicus value (2015): 58.10 DOI: 10.18535/ijecs/v6i6.19

Mining High Utility Patterns in One Phase without Generating Candidates

Rashmi Reddy M, Kavitha Juilet
Asst.Prof,RYMEC
Email Id: Rashmireddy66@gmail.com,Kavi.sayu@gmail.com

ABSTRACT

Utility mining is a new development of data mining technology. Among utility mining problems, utility mining with the itemset share framework is a hard one as no anti-monotonicity property holds with the interestingness measure. Prior works on this problem all employ a two-phase, candidate generation approach with one exception that is however inefficient and not scalable with large databases. The two-phase approach suffers from scalability issue due to the huge number of candidates. This paper proposes a novel algorithm that finds high utility patterns in a single phase without generating candidates. The novelties lie in a high utility pattern growth approach, a lookahead strategy, and a linear data structure. Concretely, our pattern growth approach is to search a reverse set enumeration tree and to prune search space by utility upper bounding. We also look ahead to identify high utility patterns without enumeration by a closure property and a singleton property. Our linear data structure enables us to compute a tight bound for powerful pruning and to directly identify high utility patterns in an efficient and scalable way, which targets the root cause with prior algorithms. Extensive experiments on sparse and dense, synthetic and real world data suggest that our algorithm is up to 1 to 3 orders of magnitude more efficient and is more scalable than the state-of-the-art algorithm

Index Terms—Data mining, utility mining, high utility patterns, frequent patterns, pattern mining

1 INTRODUCTION

FINDING interesting patterns has been an important data mining task, and has a variety of applications, for exam-ple, genome analysis, condition monitoring, cross marketing, and inventory prediction, where interestingness measures [17], [36], [41] play an important role. With frequent pattern mining [2], [3], [18], [43], a pattern is regarded as interesting if its occurrence frequency exceeds a user-specified threshold. For example, mining frequent patterns from a shopping transaction database refers to the discovery of sets of products that are frequently purchased together by customers. However, a user's interest may relate to many factors that are not necessarily expressed in terms of the occurrence frequency. For example, a supermarket manager may be interested in discovering combinations of products with high profits or revenues, which relates to the unit profits and purchased quantities of products that are not considered in frequent pattern mining.

Utility mining [41] emerged recently to address the limitation of frequent pattern mining by considering the user's expectation or goal as well as the raw data. Utility mining with the itemset share framework [19], [39], [40], for example, discovering combinations of products with high profits or revenues, is much harder than other categories of utility mining problems, for example, weighted itemset mining [10], [25], [30] and objective-oriented utility-based association mining [11], [35]. Concretely, the interestingness measures in the latter categories observe an anti-monotonic-ity property, that is, a superset of an uninteresting pattern is also uninteresting. Such a property can be employed in pruning search space, which is also the foundation of all frequent pattern mining algorithms [3]. Unfortunately, the anti-monotonicity property does not apply to utility mining with the itemset share framework [39], [40]. Therefore, utility mining with the itemset share framework is more challenging than the other categories of utility mining as well as frequent pattern mining.

Most of the prior utility mining algorithms with the item-set share framework [4], [15], [24], [29], [38], [39] adopt a two-phase, candidate generation approach, that is, first find candidates of high utility patterns in the first phase, and then scan the raw data one more time to identify high utility patterns from the candidates in the second phase.

To address the challenge, this paper proposes a new algorithm, d^2HUP, for utility mining with the itemset share framework, which employs several techniques proposed for mining frequent patterns, including exploring a regular set enumeration in a reverse lexicographic order [43] and heuristics for ordering items [18], [43]. Our contributions are as follows:

A high utility pattern growth approach is proposed, which we argue is one without candidate generation because while the two-phase, candidate generation approach employed by prior algorithms first generates high TWU patterns (candidates) with TWU being an interim, anti-monotone measure and then identifies high utility patterns from high TWU patterns, our approach directly discovers high utility patterns in a single phase without generating high TWU patterns (candidates). The strength of our approach comes from powerful pruning techniques based on tight upper bounds on utilities.

A lookahead strategy is incorporated with our approach, which tries to identify high utility patterns earlier without recursive enumeration. Such a strategy is based on a closure property and a singleton property, enhancesthe efficiency in dealing with dense data. The rest of the paper is organized as follows. Section 2 defines the utility mining problem. Section 3 surveys related works. Section 4 proposes our pattern growth approach. Section 5 presents our algorithm. Section 6 discusses the data structure and implementation. Section 7 experimen-tally evaluates our algorithm. Section 8 analyzes individual techniques. Section 9 concludes the paper.

2: UTILITY MINING PROBLEM

This section defines the utility mining problem with the itemset share framework that we study.

DOI: 10.18535/ijecs/v6i6.19

Let I be the universe of items. Let D be a database of transactions{t_1,...tn}, where each transaction ti∩I. Each item in a transaction is assigned a non-zero share. Each distinct item has a weight independent of any transaction,given by an eXternal Utility Table(XUT).The research problem of finding all high utility patterns is formally defined as follows.

Definition 1: The internal utility of an item i in a transaction t, denoted by iu(i,t), is the share of i in t. The external utility of an item i, denoted by eu(i), is the weight of i independent of any transaction. The utility of an item i in a transaction t, denoted by u(i,t),is the function f of iu(i,t) and eu(i) ,that is,u(i,t)=f(iu(i,t),eu(i)).We assume that the range of f is non-negative, that is,u(i,t)≥0.

TABLE 1

Database D and eXternal Utility Table XUT

(a) *D*: shopping transactions

TID	ITEM						
	a	b	c	d	e	f	g
t_1	1		1		1		
t_2	6	2	2			5	
t_3	1	1	1	2	6		5
t_4	3	1		4	3		
t_5	2	1		2		2	

(b) *XUT*: prices

ITEM	PRICE
a	1
b	3
c	5
d	2
e	2
f	1
g	1

Running example. Consider the data of a supermarket. Table 1a lists the quantity (share) of each product (item) in each shopping transaction where I = {a, b, c, d, e, f, g} and D = {t_1, t_2, t_3, t_4, t_5}, and Table 1b lists the price (weight) of each product. For transaction t_2 = {a, b, c, f}, we have iu(a, t_2)= 6, iu(b, t_2) = 2, iu(c, t_2) = 2, iu(f, t_2) = 5, eu(a) = 1, eu(b) = 3, eu(c) = 5, and eu(f) = 1. Here, u(i, t) is the product of iu(i, t) and eu(i). Thus, u(a, t_2) = 6, u(b, t_2) = 6, u(c; t_2) = 10, u(f, t_2Þ = 5, and so on.

Definition 2: (a) A transaction t contains a pattern X if X is a subset of t, that is, X ≥ t, which means that every item i in X has a non-zero share in t, that is, iu(i,t) ≠ 0. (b) The transaction set of a pattern X, denoted by TS(X), is the set of trans-actions that contain X. The number of transactions in TS(X) is the support of X, denoted by s(X).

Definition 3: (a) For a pattern X contained in a transaction t, that is, X ≤t, the utility of X in t, denoted by u(X,t) , is the sum of the utility of every constituent item of X in t, that is,

U(X, t) =∑u (i , t).

(b) The utility of X, denoted by u(X), is the sum of the utility of X in every transaction containing X, that is,

u(X)=∑u(X,t)=∑ u(i,t).

Definition 4: A pattern X is a high utility pattern, abbreviated as HUP, if the utility of X is no less than a user-defined mini-mum utility threshold, denoted by minU. High utility pat-tern mining is to discover all high utility patterns, that is,

HUP set = {X|X≤ I, u(X)≥ min U}.

In the running example, the manager wants to know every combination of products with sales revenue no less than 30, that is, minU = 30. Since TS({a,b}), = {t_2, t_3, t_4, t_5},we have u({a; b})= u({a, b} t_2) + u({a,b},t_3) + u({a,b}, t_4) ({a,b},t_5) = u(a, t_2) + u(b, t_2) + u(a, t_3) + u(b, t_3) + u(a, t_4) + u(b, t_4) + u(a, t_5) + u(b, t_5) + = 27. Similarly, u({a, c})= 28, u({b,c})= 24, u({a,b,c})= 31, u({a,b,c,d})= 13 and so on. Therefore, HUPset = { {a, b, c}, {a, b, d}, {a, d, e}, {a, b, d, e}, {b, d, e}, {d, e}, {a, b, c, d, e, g} }.An observation is that the utilities of patterns are neither anti-monotone nor monoto.

3 : RELATED WORKS

High utility pattern mining problem is closely related to fre-quent pattern mining, including constraint-based mining. In this section, we briefly review prior works both on fre-quent pattern mining and on utility mining, and discuss how our work connects to and differs from the prior works.

3.1:Frequent Pattern Mining

Frequent pattern mining was first proposed by Agrawal et al. [2], which is to discover all patterns whose supports are no less than a user-defined minimum support threshold. Fre-quent pattern mining employs the anti-monotonicity property: the support of a superset of a pattern is no more than the support of the pattern. Algorithms for mining frequent patterns as well as algorithms for mining high utility pat-terns fall into three categories, breadth-first search, depth-first search, and hybrid search.

This paper adopts a depth-first strategy since breadth-first search is typically more memory-intensive and more likely to exhaust main memory and thus slower. Concretely, our algorithm depth-first searches a reverse set enumeration tree, which can be thought of as exploring a regular set enumeration tree [1], [18], [33] right-to-left in a reverse lexicographic order [43]. While Eclat [43] also explores such an order, our algorithm is the first fully exploiting the benefit in mining high utility patterns.

3.2:Constraint-Based Mining

Constraint-based mining is a milestone in evolving from frequent pattern mining to utility mining. Works on this area mainly focus on how to push constraints into frequent pat-tern mining algorithms.

Pei et al. [32] discussed constraints that are similar to (normalized) weighted supports [10], and first observed an interesting property, called convertible antimonotonicity, by arranging the items in weight-descending order. The authors demonstrated how to push them into the FP-growth algorithm [18] .

Bucila et al. [9] considered mining patterns that satisfy a conjunction of anti-monotone and monotone constraints, and proposed an algorithm, DualMiner, that efficiently prunes its search space using both anti-monotone and monotone constraints.Bonchi et al. [6] introduced the ExAnte property which states that any transaction that does not satisfy the given monotone constraint can be removed from the input data-base, and integrated the property with Apriori-style algo-rithms. Bonchi and Goethals [7] applied the ExAnte property with the FP-growth algorithm. Bonchi and Lucchese [8] gen-eralized the data reduction technique to a unified framework.

De Raedt et al. [14] investigated how standard constraint programming techniques can be applied to constraint-based mining problems with constraints that are monotone, anti-monotone, and convertible.Bayardo and Agrawal [5], and Morishita and Sese [31] proposed techniques of pruning based on upper bounds when the constraint is neither monotone, anti-monotone, nor

convertible. This paper also employs such a standard technique. Our contribution is to develop tight upper bounds on the utility.

3.3 Some Categories of Utility Mining:

Interestingness measures can be classified as objective measures, subjective measures, and semantic measures [17]. Objective measures [20], [37], such as support or confidence, are based only on data; Subjective measures [13], [36], such as unexpectedness or novelty, take into account the user's domain knowledge; Semantic measures [41], also known as utilities, consider the data as well as the user's expectation. Below, we discuss three categories in detail. Yao et al. [39], [40] proposed a utility measure equivalent to Definition 3 that instantiates this framework. This paper falls into that category.

Cai et al. [10] proposed weighted itemset mining. Lin et al. [25] proposed value added association mining. Both works assigns each item a weight representing its impor-tance, which results in (normalized) weighted supports, also known as horizontal weights. Lu et al. [30] proposed to assign a weight to each transaction representing the significance of the transaction, also known as vertical weights.

Shen et al. [35] and Chan et al. [11] proposed objective oriented utility-based association mining that explicitly models associations of a specific form "Pattern ! Objective" where Pattern is a set of nonobjective-attribute value pairs, and Objective is a logic expression asserting objective-attributes with each objective-attribute value satisfying (violating) Objective assigned a positive (negative) utility.

3.4:Algorithm With the Itemset Share Framework:

As the utility measure with the itemset share framework is neither anti-monotone, monotone, nor convertible, most prior algorithms resort to an interim measure, (TWU), pro-posed by Liu et al. [29], and adopt a two-phase, candidate generation approach.

Transaction weighted utilization of a pattern is the sum of the transaction utilities of all the transactions containing the pattern. For the running example, TWU({a, b}) = 88, the sum of the utilities of transactions t_2, t_3, t_4, and t_5, TWU ({a, b, c})= 57, that of t_2 and t_3, and TWU({a, b, c, d})= 30, that of t_3. Clearly, TWU is anti-monotone.

TWU or its variants is employed by most prior algorithms, which first invoke either Apriori [3] or FP-growth [18] to find high TWU patterns (candidates), and then scan the raw data once more to identify high utility patterns from the candidates. An exception is that Yao et al. [40], [41]presented an upper bound property, that is, the utility of a size-k pattern is no more than the average utility of its size-(k-1) subsets, which is however looser than the TWU property.

Liu et al. [29] proposed the anti-monotonicity property with TWU, based on which they developed the TwoPhase algorithm by adapting Apriori [3].

Lan et al. [23] proposed an projection-based algorithm, based on the TWU model [29], that speeds up the execution by an indexing mechanism.

DOI: 10.18535/ijecs/v6i6.19

Erwin et al. [15] proposed the CTU-PROL algorithm for mining high utility patterns that integrates the TWU anti-monotonicity property and pattern growth approach [18] in the first phase, which is facilitated by a compact utility pat-tern tree structure, CUP-tree.

Tseng et al. [38] proposed the latest, FP-growth based algorithm, UP-Growth, which uses an UP-tree to maintain the revised TWU information, improves the TWU property based pruning, and thus generates fewer candidates in the first phase.

Yun et al. [42] and Dawar and Goyal [12] improved UP-Growth [38] by pruning more candidates, while the inherent issue of the two-phase approach remains.

Fournier-Viger et al. [16] improved HUIMiner [28] by pre-computing the TWUs of pairs of items to reduce the number of join operations. Krishnamoorthy [22] improved HUIMiner [28] by a partition strategy. Their improvement is within a factor of 2 to 6, while our algorithm is up to 45 times faster than HUIMiner [28] on the same databases .This paper has enhanced our preliminary work [27] with efficient computation by pseudo projection, and with optimizations by implementation. Moreover, comparative experiments with state-of-art algorithms and experimental anatomy of our individual techniques have been performed.

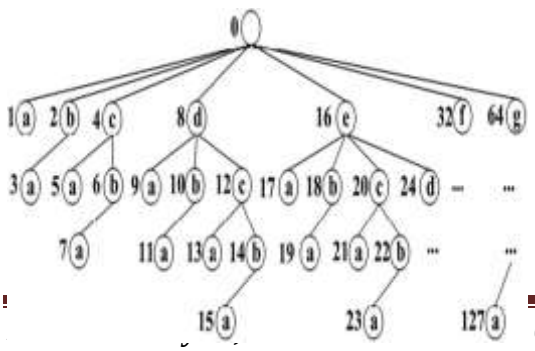


Fig.1. Reverse set enumeration tree where each node is numbered in the order of depth-first search.

4:HIGH UTILITY PATTERN GROWTH

The general approach to mining high utility pattern is to enumerate each subset X of I, and test if X has a utility over the threshold. However, an exhaustive enumeration is infeasible due to the huge number of subsets of I, and hence it is critical to employ strong pruning techniques.

This section proposes a new approach to the problem, that is, a high utility pattern growth approach. We first introduce a reverse set enumeration tree as a way to enumerate pat-terns, and then propose strong pruning techniques that drastically reduces the number of patterns to be enumerated, which lays the theoretical foundation for our algorithm.

4.1 GROWING REVERSE SET ENUMERATION TREE

Our pattern growth approach can be thought of as growing or searching a reverse set enumeration tree in a depth-first manner as shown in Fig. 1.

The construction of the reverse set enumeration tree follows an imposed ordering Ω of items. Concretely, the root is labelled by no item, each node N other than the root is labelled by an item, denoted by item(N), the path from N to the root represents a pattern, denoted by pat(N), and the child nodes of N are labelled by items listed before item(N) in Ω.

Definition 5: The imposed ordering of items, denoted by Ω, is a pre-determined, ordered sequence of all the items in I. Accordingly, for items i and j, i α j denotes that i is listed before j; i _α X denotes that i α j for every j € X, and W α X denotes that i α X for every i € W, in accordance with Ω.

The imposed ordering Ω of items can be determined by a heuristic proposed by [43]. Given Ω, a pattern can also be represented as an ordered sequence. For brevity, we use the set notation, for example, {a, b, c}, in place of the sequence notation, for example, < a, b, c > . For example in Fig. 1, the imposed ordering is the lexicographic order, i.e., Ω={a, b, c, d, e, f, g}, then a α b, a α c, a α {b, c}, {a, b}α {c, d}, and so on.

Most importantly, by such a construction, the transaction set supporting the enumerated pattern can be determined by a pseudo projection, for example, TS({a,b}) can be pro-jected from TS({b}) without materialization, and thus we can compute the utility of the pattern and a utility upper bound used for pruning in an efficient and scalable way.

4.2PRUNING BY UTILITY UPPER BOUNDING

It is computationally infeasible to enumerate all patterns, and a standard technique is to prune the search space. However, for utility mining with the itemset share framework, no anti-monotonicity property can be employed for pruning. An alternative is pruning based on utility upper bound-ing [5], [31].

With our pattern growth approach, it is to estimate

$$uB_{fpe}(X)=\sum u(fpe(X, t), t)\geq u(Y). \quad (1)$$

an upper bound on utilitie nodes in the subtree rooted at the node currently being explored, when growing the reverse set enumeration tree. subtree can be pruned as all patterns in the subtree are not high utility patterns.

Definition 6: Given an ordering Ω, a pattern Y is a prefix extension of a pattern X, if X is a suffix of Y , that is if Y = W ∩ X for some W with W α X in Ω.

Definition 7: Given an ordering Ω, a pattern Y is the full prefix extension of a pattern X w.r.t. a transaction t containing X, denoted as Y = fpe(X, t), if Y is a prefix extension of X derived by adding exactly all the items in t that are listed before X in Ω, that is, if Y = W ∩ X with W={i|i€t∩iαX∩X≤t}.

For the running example, the full prefix extensions of {c} w.r.t. t_1 and t_2 are fpe({c}, t_1) = {a,c} and fpe({c}, t_2) = {a, b, c} respectively.

Theorem 1 (Basic upper bounds). For a pattern X, the sum of the utility of the full prefix extension of X w.r.t. each transaction in TS(X), denoted by $uB_{fpe}(X)$, is no less than the utility of any prefix extension Y of X, that is,

Proof. The premise, Y is a prefix extension of X, means X ≤ Y , and thus has two implications. First, TS(Y) _TS(X).Second, ¥t € TS(Y), Y ≤fpe(X, t). As the utility function is nonnegative, we haveminU, and so on. Thus, Nodes 1 and 2

DOI: 10.18535/ijecs/v6i6.19

(with Node 3) are pruned, and Nodes 4, 8, 16, 32, and 64 will be visited.

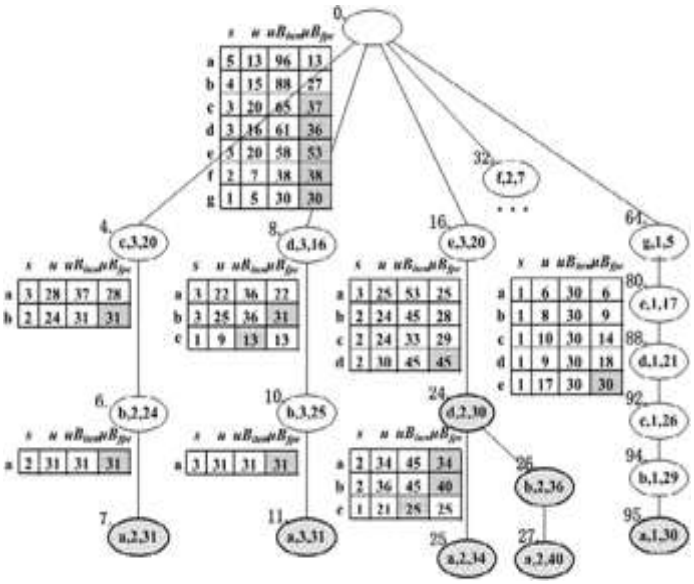
5 Mining Patterns In One Phase Without Candidate Generation

This section presents our algorithm, d^{2}HUP, namely Direct Discovery of High Utility Patterns, which is an reduces the number of patterns to be enumerated, and a novel data structure that enables efficient computation of utilitiesw and upper bounds which will be detailed in section 6.1.

Moreover ,our algorithm lists items in the descending order of uBitem based on a heuristic proposed by[43].The pseudo code of d2HUP is shown in Algorithm 1,which works as follows.

Algorithm 1.d2HUP (D, XUT, minU)

```
1   build TS({}) and Ω from D and XUT
2   N<-root of reverse set enumeration tree
3   DFS(N,TS(pat(N)),minU,Ω
    Subroutine: DFS (N,TS (pat,N)),minU,Ω)
4   if u(pat(N))≥minU then output pat(N)
5   W<-{i|iαpat(N)∩uBitem(I,pat(N))≥minU}
6   If Closure(pat(N),W,minU) is satisfied
7   Then output nonempty subset of W∩pat(N)
8   Else if Singleton(pat(N),W,minU) is satisfied
9   Them output W∩pat(N) as an HUP
10  Else foreach item i€W in Ω do
11  If uNfpe({i}∩pat(N))≥minU
12  Then C<- the child node of N for i
13  TS(pat(C))<- Projects (TS(pat(N)),i)
14  DFS(C,TS(pat(C)), minU , Ω)
15  End foreach
```



5.1:Revisit the Running Example

The execution process of d^{2}HUP can be thought of as searching a pruned version of a reverse set enumeration tree, which is shown in Fig. 2 for our running example where each node N is labelled with item(N), s(pat(N)), and u(pat(N)).

Such a summary table is also attached to N in Fig. 2.

When visiting Node 4, the utilities and bounds for i €{a,b} are already maintained in TS({c}) which is derived from TS({}) by a pseudo projection presented in Section 6.3.

It turns out that {c} represented by Node 4 is not a high util-ity pattern, neither the closure property nor the singleton property holds, and Node 5 is pruned as $uB_{fpe}(\{a, c\})<minU$. Subsequently, DFS will recursively visits Node 6 where the closure property holds and hence {a, b, c} is out-put as a high utility pattern without visiting Node 7.

The remaining nodes that will be explored in the order of depth-first search are Node

8, Node 10 where the closure property holds, Node 16, Node 24 where the closure prop-erty

also holds, Node 32, and Node 64 where the singleton property holds and hence {a, b, c, d, e, g} is identified as the only high utility pattern without searching the subtree under Node 64.

In short, d^2HUP only enumerates Nodes 0, 4, 6, 8, 10, 16, 24, 32, and 64, a total of nine nodes, in finding all the high utility patterns, while the entire reverse set enumeration tree consists of 2^7 ¼ 128 nodes.

6:EFFICIENT IMPLEMENTATION BY REPRESENTING TRANSACTIONS SCALABLY

When growing the reverse set enumeration tree, the d^2HUP algorithm needs to determine TS(pat(N)) for each node N being visited for computing utilities and utility upper bounds for prefix extensions of pat(N) as shown at line 1 and line 13 in Algorithm 1. How to represent and maintain TS(pat(N)) together with related utilities and upper bounds is the key to the scalability and efficiency of the proposed algorithm.

This section introduces a linear data structure, CAUL, namely a Chain of Accurate Utility Lists, which is not tree-based, nor graph-based, but simply consists of linear lists. CAUL maintains the original utility information for each enumerated pattern in a way that enables us to compute the utility and to estimate tight utility upper bounds efficiently.

6.1:Scalable Representation of Utility Information

For the pattern, pat(N), represented by a reverse set enumeration tree node N currently visited by a depth-first search, we use CAUL to maintain the utility information in the transaction set TS(pat(N)) of the node N, denoted by $TS_{caul}(pat(N))$, which is necessary for computing the utilities and upper bounds of its prefix extensions. $TS_{caul}(pat(N))$ consists of two parts, utility lists and a summary table.

For each transaction t € TS(pat(N)), there is a utility list holding the utilities of all the items in t relevant in growing prefix extensions of pat(N). That is, In addition, an extra element is appended to the utility list to maintain u(pat(N), t).

The summary table maintains an entry for each distinct item j relevant in growing prefix extensions of pat(N), which is denoted as a quintuple, summary[j]& = (s[j];u[j]; $uB_{item}[j]$; $uB_{fpe}[j]$; link[j]), as described in the following. Summary

entries are also arranged in the imposed ordering Ω .

TScaul({}) is built by scanning the database d and the external utility table XUT, filtering out globally irrelevant items ,and computing s[j],u[j],uBitem[j],and uB

For example, Fig 3 shows TScaul({}) for Node 0 in Figs1 and 2. The first list representation t1 with its first element storing item a and u(a,t1), its second element storing item c and u(c,t1)=5, and so on .In any of the five lists, there is no extra element to hold the utility of{} in the transaction since it is 0. The occurrences of item a in all the five lists are threaded by link[a] of the first summary entry. The other components , s[a], u[a],uBitem[a],and uBfpe[a],of the first summary entry keep s({a}),u({a}),uBitem(a,{}),and uBitem({a}) respectively

DOI: 10.18535/ijecs/v6i6.19

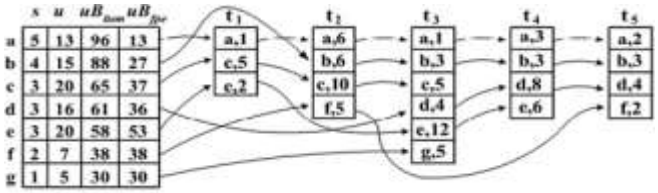


Fig. 3.TS_{caul}ðfgÞ: CAUL representing transaction set TSðfgÞ, derived from D in Table 1, for the null root of the reverse set enumeration trees in Figs. 1 and 2.

6.2: Approach Generating No Candidates Enabled

One difference between our CAUL and the data structures by prior algorithm [4],[15],[24],[29],[38] is that CAUL keeps the original utility information for each transaction, while the latter keep the utility estimate, TWU, instead. This is the root cause why we are able to mine high utility patterns without generating candidates ,while the prior algorithms have to take a two-phase, candidate generation approach.

6.3: Computation by Pseudo Projection

For any node N and its parent node P with pat(N) =(i) ∩ pat(P) on the reverse set enumeration tree, TS_{caul}(pat(N)) can be efficiently computed by a pseudo projection [26], where the pseudo TS_{caul}(pat(N)) shares the same memory space with TS_{caul}(pat(P).

projection [26], where the pseudo TS_{caul}(pat(N)) shares the same memory space with TS_{caul}(pat(P).

The utility lists of the pseudo TS_{caul}(pat(N)) are delimited by following link[i] in TS_{caul}(pat(P)), and the summary entry for each item j α i of the pseudo TS_{caul}(pat(N)) is computed by scanning each delimited utility list.

Algorithm 2. PseudoProject (TScaul (pat (P)),i)

```
1  foreach relevant item jαi do
2  (s[j],u[j],uBitem[j],uBfpe[j],link[j])<-0
3  End foreach
4  Foreach utility list t threaded by link[i] do
5  U(pat(N),t)<-u(pat(P)+u(I,t)
6  ∑<-u(pat(N),t)
7  Foreach relevant item j € t ∩ j α i by Ω do
8  s[j] <- s[j]+1
9  u[j] <- u[j]+u(j,t)+u(pat(N),t)
10 ∑<- ∑+u(j,t)
11 uBfpe[j]<-uBfpe[j]+∑
12 end foreach
13 foreach relevant item j € t ∩ j α i by Ω do
14 uBitem[j] <- uBitem[j]+∑
15 thread t into the chain by link[j]
16 end foreach
17 end foreach
```

6.4 :MATERIALIZATION VERSUS PSEUDO PROJECTION

We may only keep TS_{caul}({}) in memory and get TS_{caul} for all patterns

by recursive pseudo projection, which is

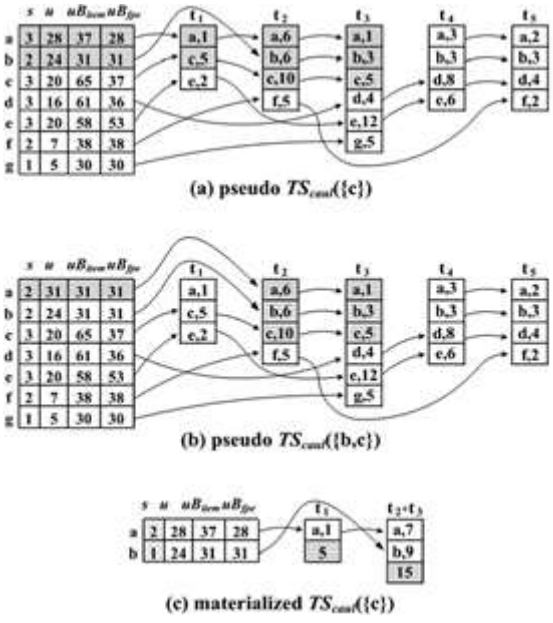


Fig. 4.Projections from TS_{caul}ðfgÞ.

The most scalable way to maintain original utility information and is the core of our approach. Furthermore, we can optimize our approach by considering two trade-offs.

Maximum number of rounds g for irrelevant item filtering. The body of the Pseudo Project algorithm iterates multiple rounds to filter out irrelevant items. We introduce a parameter, g, to make a tradeoff between the benefit of mpruning by tightening upper bounds and the additional computational overheaiterative irrelevant item filter-ing. When no more irrelevant items are identified by Corol-lary 2 or a maximum number of rounds has been reached, the iteration terminates.

Materialization threshold f for space-time tradeoff. We introduce a materialization threshold f to make a tradeoff between the scalability resulted from representing TS_{caul}(pat(N)) by pseudo projection and the efficiency resulted from leaving out irrelevant items by materializing TS_{caul}(pat(N)). When the percentage of relevant items is below the threshold, a materialized copy will be made by copying the pseudo TS_{caul}(pat(N)) to memory space separate from TS_{caul}(pat)))

For example, Fig. 4c shows the materialized TS_{caul}({c}) where the summary entries are copied from 4a, the first list has an element for the only relevant item, a, in t_1, and a special element for u({c},t_1). As the sets relevant items are identical, t_2 and t_3 are merged into the second list.

7:COMPARATIVE EVALUATION

We evaluate our d^2HUP algorithm by comparing with the state-of-the-art algorithms, TwoPhase [29], $IHUP_{TWU}$ [4], UP-Growth [38], and HUIMiner [28]. The code of

TABLE 2

Characteristics of Six Datasets

Dataset	Jtj	jIj	jDj	Type
T10I6D1M	10 : 33	1,000	933;493	mixed
WebView-1	2:5 : 267	497	59;602	sparse
Chess	37 : 37	76	3;197	dense
Chain-store	7:2 : 170	46,086	1;112;949	sparse
T20I6D1M	20 : 49	1,000	999;287	mixed
Foodmart	4:8 : 27	1,559	34;015	dense

TwoPhase [29] and HUIMiner [28] were provided by the original authors. Due to unavailability, we implemented an improved version of $IHUP_{TWU}$ [4] and an improved version of UP-Growth [38], namely $IHUP^{þ}_{TWU}$ and $UP^{þ}_{UPG}$ respectively. The latter employ a search tree to compactly represent all candidates, facilitate fast matching between candidates and transactions, and improve the efficiency of the second phase greatly. When mining large databases, $UP^{þ}_{UPG}$ is even faster than HUIMiner [28], and UP-Growth [38] simply did not report the running time of the second phase because it is too long [38].

Six datasets are used in comparative experiments. T10I6D1M and T20I6D1M with utility information are exactly the same dataset as in [29], and Chain-store is the same as in [4], [28], [29], [38]. WebView-1 and Chess contain no utility information originally. We generate the

DOI: 10.18535/ijecs/v6i6.19

utility information by following the method in [29]. So do [28], [38]. Thus, Chess with utility information used by [28], [38] and us share the same features, but are not the same datasets. Foodmart is from the Microsoft foodmart database. The datasets are sum-marized by Table 2 where the first column is the name of a dataset, the second (|t|) is the average and maximum length of transactions, the third (|I|) is the number of distinct items, the fourth (|D|) is the number of transactions, and the fifth (Type) is a rough categorization based on the number of high utility patterns to be mined, partially depending on the mini-mum utility threshold as in Table 3.

The minimum utility thresholds minU(percent) in terms of the percentage of overall utility for each dataset are selected in a way that the results can be verified with [4], [28], [29], [38]. The experiments were performed on a PC with 1.80 GHz CPU and 8 GB memory running CentOS 6.3. The parameter setting of g = 3 and f = 0:5 is used as the default for d^2HUP unless specified otherwise, which is discussed in Section 8.1.

CAUL enables tightening upper bounds iteratively in the mining process while the data structures by $UP^{\flat}_{UPG}$, $IHUP^{\flat}_{TWU}$, and TwoPhase cannot.

7.1:Enumerated Patterns and Candidates

Table3 shows the number of high utility patterns(hups), the maximum length utility patterns(ml),the numbers of patterns enumerated by our d2HUP algorithm and HUI-Miner respectively, and the numbers of candidates generated in the first phase by the TwoPhase respectively, with different datasets and varying minU.

7.2:Running Time and Memory Usage

Fig. 5 shows the running time by the five algorithms. For example, for T10I6D1M with minU¼ 0:01%, d^2HUP takes 27 seconds, HUIMiner 154, $UP^{\flat}_{UPG}$ 101, $IHUP^{\flat}_{TWU}$ 109, and Two-Phase runs out of memory. The observations are as follows.

First, d^2HUP is up to 1 to 3 orders of magnitude more efficient than $UP^{\flat}_{UPG}$, $IHUP^{\flat}_{TWU}$, and TwoPhase. In particular,

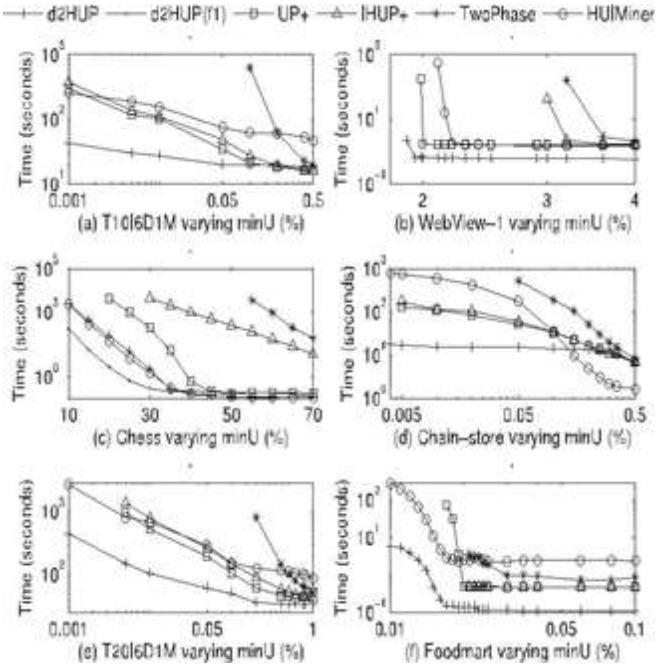


Fig. 5.Running time versus minU (percent).

d^2HUP is up to 6.6, 6.8, 7.8, 261, 472, and 1,502 times faster than $UP^{\flat}_{UPG}$ on T20I6D1M, T10I6D1M, Chain-store, Web-View

DOI: 10.18535/ijecs/v6i6.19

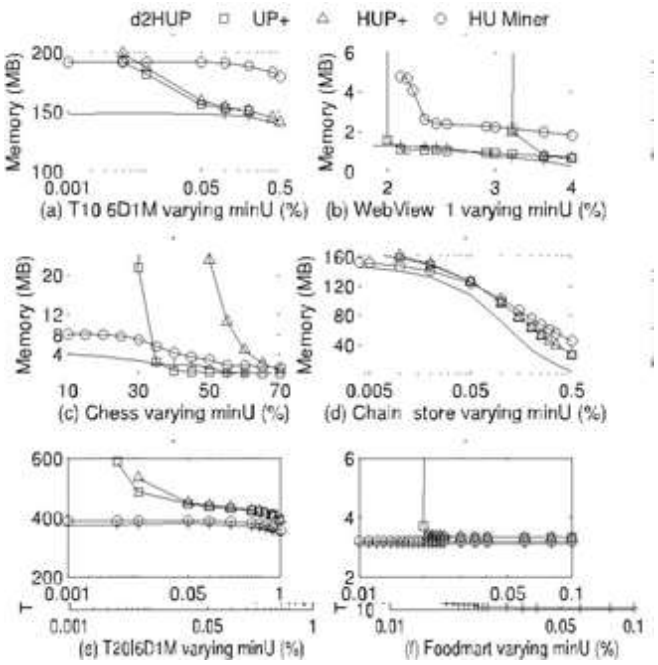


Fig. 6.Running time with varying data characteristic.

for d^{2}HUP is to make a materialized copy of the pseudo CAUL (f ¼ 1).

Memory usage. We collect the peak memory usage statistics by every algorithm during its execution except TwoPhase as shown in Fig. 6. For example, for T10I6D1M with minU = 0:1%, the peak memory usage by d^{2}HUP is

147 MB, and that by HUIMiner, by UP$^{þ}_{UPG}$, and by IHUP$^{þ}_{TWU}$ are 191, 153, and 154 MB respectively. The fol Our d^{2}HUP algorithm uses the least amount of mem-ory because d^{2}HUP uses CAUL that is more compact than the vertical data structure by HUIMiner, and d^{2}HUP does not materialize candidates in memory while UP$^{þ}_{UPG}$, IHUP$^{þ}_{TWU}$, and TwoPhase do.

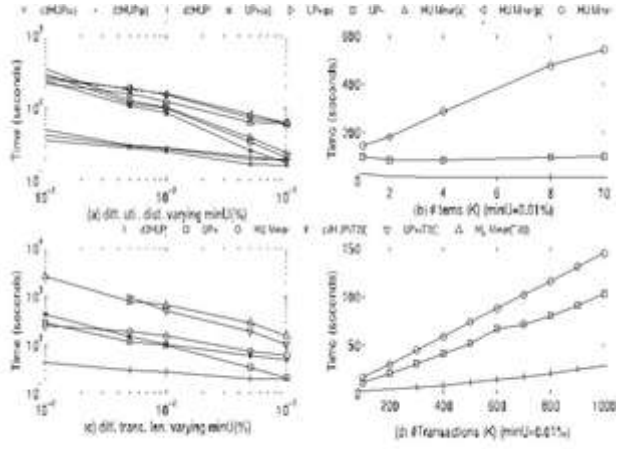


Fig. 7.Peak memory usage versus minU (percent).

Our d^{2}HUP algorithm uses the least amount of mem-ory because d^{2}HUP uses CAUL that is more compact than the vertical data structure by HUIMiner, and d^{2}HUP does not materialize candidates in memory while UP$^{þ}_{UPG}$, IHUP$^{þ}_{TWU}$, and TwoPhase do.

The memory usage by UP$^{þ}_{UPG}$ and IHUP$^{þ}_{TWU}$ are 50 percent to 2 orders, and 90 percent to 2 orders of magnitude more than d^{2}HUP respectively. Two-Phase uses the most, and usually runs out of mem-ory when minU is small.

Comparison with Varying Data Characteristics

7.2:Comparsion with Varying Data Characteristics

We compare our d^{2}HUP algorithm with the best prior algo-rithms, HUIMiner and UP$^{þ}_{UPG}$ on varying data characteris-tics, including different utility distributions, changing number of items, different average length of transactions, and changing data size based on the T10I6D1M dataset as it is large and of a mixed type.

First, we generate external utilities anti-proportional to supports and proportional to supports, in addition to generating external utilities randomly. Fig. 7a shows the running time

DOI: 10.18535/ijecs/v6i6.19

of the three algorithms on the respective resulting data-set with minU ranging from 0.1 percent down to 0.001 per-cent. The running time with external utilities anti-proportional to supports, as depicted by '(a)', is less than that proportional to supports, as depicted by '(p)', and the latter is less than that generated randomly. For every utility distribution, d^2HUP takes much less time than HUIMiner and$UP^{þ}_{UPG}$.

Second, we conduct a comparative experiment with the number of items ranging from 1K to 10K as shown inFig. 7b. The running time by d^2HUP and by $UP^{þ}_{UPG}$ do not change much because relevant items do not increase muchwith the increase of items. However, the running time by HUIMiner increases sharply with the increase of items.

Third, we evaluate the effect of the transaction lengths by comparing results both on T10I6D1M and on T20I6D1M. As in Fig. 7c where the results with T20I6D1M are depicted by '(T20)', the running time increases with the average length of transactions since both the average length and the num-ber of high utility patterns also increase, so do the running time gaps among d^2HUP, HUIMiner, and $UP^{þ}_{UPG}$.

Finally, Fig. 7d shows the scalability evaluation result with jDj varying from 100K to 1000K. Clearly, d^2HUP has better scalability than HUIMiner and $UP^{þ}_{UPG}$ according to the slopes of the curves.

8:EXPERIMENTAL ANATOMY OF D^2HUP

8.1:Analysis of Additional Pruning Techniques

First of all, let us note that our "basic approach" is to depth-first search the reverse set enumeration tree with pruning by basic upper bounding (Theorem 1) which is enabled by the pseudo projection of CAUL. In terms of the maximum number of rounds g for iterative irrelevant item filtering and the materialization threshold f, our "basic approach" corresponds to the setting of g ¼ 1 and f ¼ 0 without lookahead.

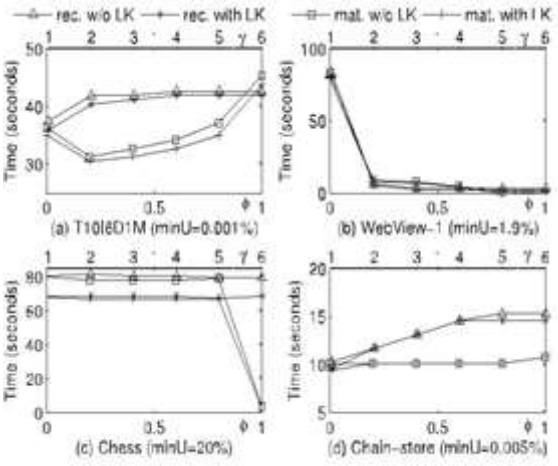


Fig. 8 Running time of d^2HUP with varying g and f reports the running time with g ranging from 1 to 6 and f ranging from 0 to 1 both with and without lookahead. By comparing with Fig. 5 we can find that our "basic approach" already outperforms prior algorithms signifi-cantly. For example, for the T10I6D1M dataset (minU = 0:001%).

DOI: 10.18535/ijecs/v6i6.19

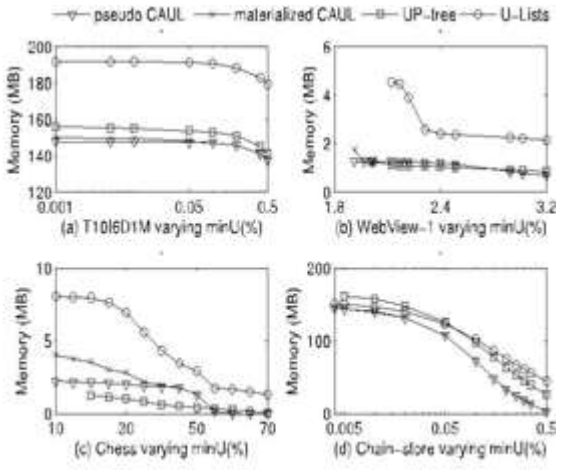


Fig. 9.Memory footprints of trans. representations

Third, in terms of pruning the search space, more irrelevant item filtering (Corollaries 2 and 3) by increasing g, as depicted by 'rec', and more CAUL materialization by increasing f, as depicted by 'mat', are very helpful as the utility upper bounds become tighter, which also decreases the running time with sparse data, for example, for WebView-1. However, it comes with additional computational overhead and thus the running time does not always decrease with the increase of g and f, for example, for Chain-store.In short, while the setting of g = 1 and f = 0 with look-ahead is good, we recommend to use the setting of g = 3 and f = 0:5 with lookahead as the default. Evaluating the Transaction Representation We evaluate the memory footprints of thepseudo CAUL and the materialized CAUL by d^2HUP, the memory footprint ofthe tree based structure, UP-tree [38].

Third, U-List has the largest memory footprint because U-List has no compression at all.Finally, materializing CAUL usually increases the mem-ory footprint by a small percentage, but by a large percent-age for a dense dataset, like Chess. In the latter case, the overall memory footprint may still be small as a dense data-set is usually not that large.

9:CONCLUSION AND FUTURE WORK

This paper proposes a new algorithm, d^2HUP, for utility mining with the itemset share framework, which finds high utility patterns without candidate generation. Our contributions include: 1) A linear data structure, CAUL, is proposed, which targets the root cause of the two-phase, candidate generation approach adopted by prior algorithms, that is, their data structures cannot keep the original utility information. 2) A high utility pattern growth approach is presented, which integrates a pattern enumeration strategy, pruning by utility upper bound-ing, and CAUL. This basic approach outperforms prior algorithms strikingly. 3) Our approach is enhanced sig-nificantly by the lookahead strategy that identifies high utility patterns without enumeration.

In the future, we will work on high utility sequential pat-tern mining, parallel and distributed algorithms, and their application in big data analytics.

REFERENCES :

R. Agarwal, C. Aggarwal, and V. Prasad, "Depth first generation of long patterns," in Proc. ACM SIGKDD Int. Conf. Knowl. Discov-ery Data Mining, 2000, pp. 108–118.

R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in Proc. ACM SIG-MOD Int. Conf. Manage. Data, 1993, pp. 207–216.

R. Agrawal and R. Srikant, "Fast algorithms for mining associa-tion rules," in Proc. 20th Int. Conf. Very Large Databases, 1994, pp. 487–499.

C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and Y.-K. Lee, "Efficient tree structures for high utility pattern mining in incremental data-bases," IEEE Trans. Knowl. Data Eng., vol. 21, no. 12, pp. 1708– 1721, Dec. 2009.

R. Bayardo and R. Agrawal, "Mining the most interesting rules," in Proc. 5th ACM